Université de Technologie de Compiègne
Génie Informatique

Étudiant : **Erwan Normand**
Suiveur UTC : **Mohamed Sallak**
Semestre : **Automne 14**

Amélioration et conception de workflows

Entreprise : **VideoStitch**
Lieu : **Paris**
Responsable : **Nicolas Burtey**

Avant-propos

Remerciements

Je souhaite remercier tout d'abord l'UTC et VideoStitch pour m'avoir permis de réaliser ce stage technique d'assistant ingénieur. Ces 7 mois très riches m'ont permis de constituer une première expérience technique et professionnelle solide en ingénierie logicielle. Ce stage fut une transition pour moi de la première année commune du Génie Informatique de l'UTC à la filière de spécialisation d'Ingénierie des Connaisances et des Systèmes d'Informations.

Je remercie sincèrement Nicolas Burtey, pour m'avoir accueilli dans son entreprise, pour la confiance qu'il m'a témoignée en acceptant ma candidature ainsi que pour son suivi régulier de mon stage et du temps qu'il a consacré à mes nombreuses questions sur l'entrepreneuriat.
Je remercie en particulier Andrés Peralta, Nicolas Lopez et Julien Fond avec qui j'ai le plus souvent travaillé, pour leurs aides et leurs nombreux conseils.
Je remercie au même titre toute l'équipe, ingénieurs, commerciaux, et stagiaires de VideoStitch avec qui j'ai pu longuement travailler : Henri Rebecq, Nils Duval, Rodolphe Fouquet, Wieland Morgenstern, Raphaël Lemoine, Jean Vittor, Aksel Piran, Liesbeth De Mey, Florent Melchior, Rodolphe Chartier, Alexis Pontin ainsi que les deux UTCéens avec qui j'ai effectué mon stage : Marie Chatelin et Jean Duthon. Merci à tous pour leurs aides, les nombreux partages et connaissances échangés. Venir travailler fut un plaisir et une motivation tous les jours grâce à eux tous et à l'excellente ambiance dans l'équipe.

Enfin, je remercie toutes les personnes qui m'ont consacré de leur temps au support de leurs produits, et merci à ceux qui les ont conçus. Notre travail est entouré d'outils qui innovent sans cesse et où nous prenons notre place. Il est bon de se rappeler que toute innovation a ses fondations ; et j'espère que les briques que j'ai apportées seront la source de nouvelles idées.

Résumé

J'ai réalisé un stage de 7 mois dans la start-up VideoStitch, basée sur Paris et qui conçoit des logiciels de vidéo panoramique à 360°. Au sein d'une équipe utilisant la méthode Scrum, j'ai réalisé deux missions d'amélioration et de conception de *workflows* à l'usage de cette équipe : d'une part j'ai simplifié et fait évoluer le *workflow* de développement vers une gestion multi-logicielle tout en intégrant deux nouveaux produits ; d'autre part, j'ai développé et déployé un système de *plugins* augmentant les capacités entrées/-sorties des logiciels de l'entreprise. Sur ces deux projets, il s'agissait de trouver le juste milieu entre réaliser le *workflow* le plus complet et simple, quitte à factoriser des solutions déjà existantes, et réaliser *workflow* le plus rapide à déployer, permettant de le développement en tant que tel des produits. À l'échelle de la start-up il s'agissait de positionner le curseur entre qualité et délai pour tenir les coûts.

Mots-clés

Génie Logiciel, *workflow*, Git, Qt, Programmation Orientée Composants, *plugin* Programmation Orientée Objet, *Sofware Development Kit*

Sommaire

Introduction

Dans le cadre de la formation ingénieur de l'Université de Technologie de Compiègne, le 3e semestre (bac+4) est consacré à un stage de 24 semaines d'assistant ingénieur. Cette période de travail se déroule dans le milieu professionnel, du secteur privé ou public, dans une optique d'une première expérience longue, qui a pour objectif une confrontation au monde du travail, de comprendre ses contraintes et enjeux, et enfin de tirer un enseignement d'une mise en application pratique des connaissances reçues à l'UTC jusqu'ici.

Mon stage s'est déroulé à VideoStitch, éditeur de logiciels de vidéo panoramique à 360°, dans les locaux de Telecom Paris Tech dans le 14ème arrondissement de Paris. Il a été précédé d'un CDD à temps partiel de 7 semaines, totalisant avec le stage une période de travail de 7 mois, du 15 juillet 2014 au 13 février 2015.

Ce rapport est consacré au compte rendu de ce travail ; plus particulièrement aux missions qui m'ont été confiées, aux difficultés rencontrées et les solutions proposées ainsi qu'aux apprentissages et bilans acquis.

1. Présentation de l'entreprise

1.1 VideoStitch

1.1.1 Présentation générale

VideoStitch est une entreprise française d'édition de logiciels, qui conçoit et vend des logiciels de capture, de montage, de montage, de diffusion et de lecture de vidéos panoramiques à 360°. Elle s'intéresse par extension au marché de la réalité virtuelle. Son site internet est à l'adresse suivante : http://www.video-stitch.com/.

FIGURE 1.1 – Logo de VideoStitch

C'est une Société à Actions Simplifiées (SAS) basée sur Paris, qui emploie moins d'une dizaine de personnes actuellement. La plupart ayant été embauchés il y a quelques mois, c'est une société en forte croissance. Nicolas Burtey en est le président.

1.1.2 L'équipe

L'équipe est actuellement composée de 8 personnes et d'un nombre variable de 2 à 5 stagiaires.

Nicolas Burtey et Aksel Piran sont en charge de la partie commerciale et marketing de l'entreprise. Ils consacrent également du temps dans la recherche d'investisseurs et la présentation de l'entreprise dans des salons ou des conférences. En outre, M. Burtey suit l'avancement des produits en passant un jour ou deux dans la semaine avec l'équipe des développeurs.

Nicolas Lopez, Wieland Morgenstern et Henri Rebecq travaillent sur le cœur du projet, VideoStitch SDK (section 1.3.4 p.13), mettant au point algorithmes et nouvelles fonctionnalités.

Andrés Peralta, Julien Fond et Raphaël Lemoine travaillent quant à eux à la conception des trois autres produits de VideoStitch basés sur VideoStitch SDK : VideoStitch Studio (section 1.3.1 p.10), Vahana VR (section 1.3.2 p.12), VideoStitch Player (section 1.3.3 p.13).

Liesbeth De Mey, stagiaire, est en charge de réaliser des démonstrations et des tutoriels vidéos des produits de l'entreprise. Enfin, Jean Duthon et Marie Chatelin, stagiaires UT-

Céens également en TN09, sont en charge, respectivement, de la *QA* [1] et de l'intégration continue des quatre produits pour Jean, et d'assister l'équipe au développement des logiciels pour Marie. Leurs rapports contiennent de plus amples détails sur le déroulé de leurs stages.

1.1.3 Historique

VideoStitch a été fondée par ce même Nicolas Burtey en janvier 2012.
Photographe diplômé de l'École Lumière, s'étant fait connaître par la photo panoramique et la photo fish-eye, M. Burtey évolue par la suite dans le domaine de la vidéo panoramique.

FIGURE 1.2 – Logo de Loop'In

Il crée sa société de production vidéo, Loop'In, avec laquelle il réalise une certain nombre de vidéos panoramiques à 360°pour le compte de divers clients (publicité essentiellement). Des exemples de ces vidéos sont disponibles à cette adresse : http://www.loop-in.com/galerie/. La vidéo panoramique, étant alors un procédé très jeune encore inconnu du grand public, est lente et difficile à réaliser : beaucoup de tâches doivent être faites de manière manuelle et répétitive, avec des outils inadaptés car non prévus à cette utilisation. Si la création de photos panoramiques et à 360°est aisée, il manque toutefois un logiciel complet et mature qui permettrait l'édition et le montage de vidéos panoramiques.

M. Burtey décide alors de créer VideoStitch pour concevoir cet outil, un logiciel d'édition et de montage de vidéos panoramiques : VideoStitch Studio. Quand ce logiciel atteint une version stable, en 2014, VideoStitch oriente alors ses efforts vers un nouveau logiciel, Vahana VR, permettant cette fois ci une réalisation et une diffusion en direct de vidéos panoramiques à 360°.
Le but de la société est de permettre une production aisée et de qualité de vidéos panoramiques à 360°, de la capture des images à la diffusion de la vidéo finale, en passant par sa réalisation complète.

1.2 La vidéo 360

1.2.1 Du panorama à la vidéo 360

Une image panoramique, ou panorama, est une superposition d'images qui sont assemblées pour composer une vue bien plus large que ne le permet une image seule. La

1. *Quality Assurance*

figure 1.3 présente un exemple d'assemblage d'un panorama.
Concrètement, avec un appareil photographique, cela consiste à prendre plusieurs vues successives en tournant l'appareil autour d'un axe entre deux clichés. Les photos obtenues forment ainsi une série d'un même sujet et doivent pouvoir se recouvrir en partie pour être ensuite assemblées en une seule image panoramique. Cet assemblage peut être réalisé avec directement par l'appareil, comme c'est le cas des téléphones portables, ou par un logiciel spécialisé, comme Photoshop[1] ou PTgui[2].

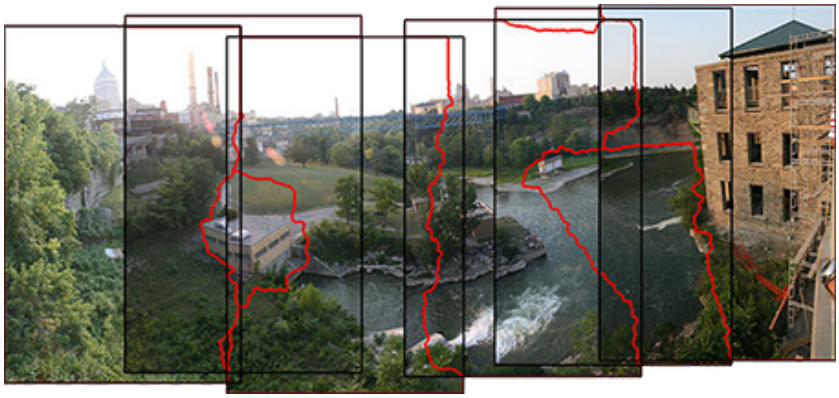

FIGURE 1.3 – Exemple d'assemblage de panorama avec détection des zones de recouvrement[3]

La vidéo panoramique consiste à filmer avec plusieurs caméras en même temps, chacune capturant une partie du panorama, pour assembler les images capturées en une seule vidéo par le même principe que la photo panoramique.
Avec suffisamment de caméras, il est possible de couvrir un angle de prise de vue de 360°à l'horizontale et de 180°à la verticale. Le panorama créé forme alors une sphère virtuelle, comme le montre la figure 1.4. C'est ce qu'on appelle la vidéo 360.

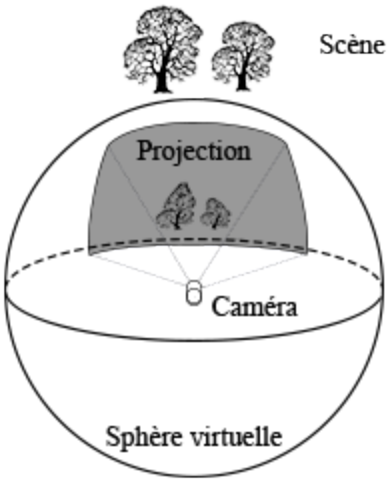

FIGURE 1.4 – Exemple d'une scène filmée avec une caméra

Cependant, comme le montre la figure 1.3, un panorama reste une image – ou une vidéo

−, c'est-à-dire une matrice plane constituée de pixels [2]. Or une vidéo 360 représentant une sphère virtuelle, ou plutôt sa surface, elle sera nécessairement une *projection* de cette sphère, c'est-à-dire une correspondance mathématique entre cette surface en 3 dimensions et cette vidéo plane[4].
Dans la vidéo 360, la projection la plus utilisée est la projection équirectangulaire[5]. Un exemple en est le planisphère terrestre, où après une projection cylindrique équirectangulaire sur la figure 1.5 le plan est déroulé pour obtenir le planisphère de la figure 1.6.

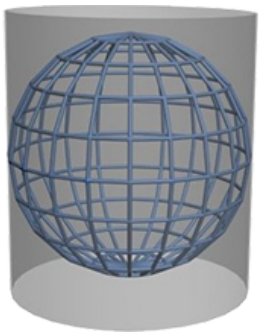

FIGURE 1.5 – Schéma d'une projection équirectangulaire[6]

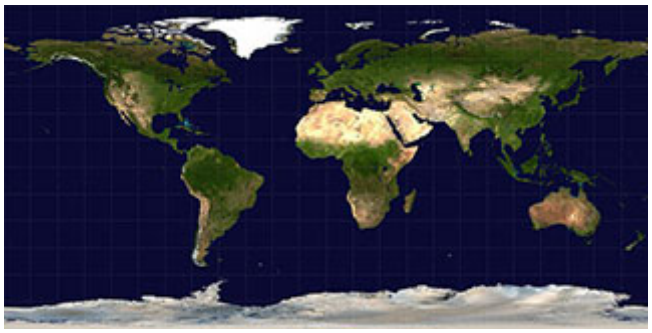

FIGURE 1.6 – Projection équirectangulaire de la Terre[7]

Un lecteur vidéo 360 est alors nécessaire pour lire correctement la vidéo : il va reconstruire la sphère, pour y projeter le panorama et y placer le spectateur au centre de cette sphère. Dès lors, le spectateur peut déplacer le regard tout autour de lui pour embrasser le panorama : on entre dans le champ de la réalité virtuelle [3].

1.2.2 Le marché de la vidéo 360

Si les années 2000 ont vu la démocratisation des appareils photographiques numériques, les années 2010 voient l'émergence des caméras numériques Haute Définition [4] bon marché, comme les GoPro, et des écrans 4K [5]. De plus les capacités informatiques grand public permettent désormais de stocker et traiter plusieurs vidéos HD en même temps.
Dès lors la vidéo 360 peut se développer, son coût technique étant accessible, et le marché, bien qu'encore restreint et méconnu du public, est un plein développement et présente déjà de très nombreux concurrents. Jaunt, entreprise américaine créée en mai 2013 a, par exemple, déjà levé 35 millions de dollars de fonds d'investissement[9].
Les grandes entreprises comme Facebook[10], Samsung[11], Microsoft[12] ou Google[13] suivent de très près ce marché, notamment pour concevoir des produits de réalité virtuelle ou de réalité augmentée, qui progressivement vont s'installer dans les usages du public,

2. Pour « picture element »
3. La réalité virtuelle est ici comprise dans son sens de « Simulation sensorielle immersive de la réalité »[8]
4. Définition d'image de 1920x1080 pixels
5. Définition d'image de 4096x2160 pixels

tout comme l'ont fait auparavant les ordinateurs portables, les téléphones mobiles et les tablettes numériques.

1.3 Produits et stratégies de l'entreprise

Par rapport à ce marché, VideoStitch se positionne aujourd'hui comme un challenger et propose actuellement quatre produits[14].

1.3.1 VideoStitch Studio

Premier logiciel de l'entreprise, sortis en version stable en 2013 pour Windows, Mac OS X et Linux, il permet une conception de vidéos 360. Le *workflow* de travail avec le logiciel est le suivant :

Importation des vidéos

Le cas le plus commun est de capturer la vidéo 360 à l'aide de 6 caméras GoPro installées sur une monture spécialisée [6] comme sur la figure 1.7. Les caméras sont configurées pour filmer en HD, et, munies de leurs optiques d'origine, six sont nécessaires pour filmer toute la sphère du panorama.

FIGURE 1.7 – Une monture 360

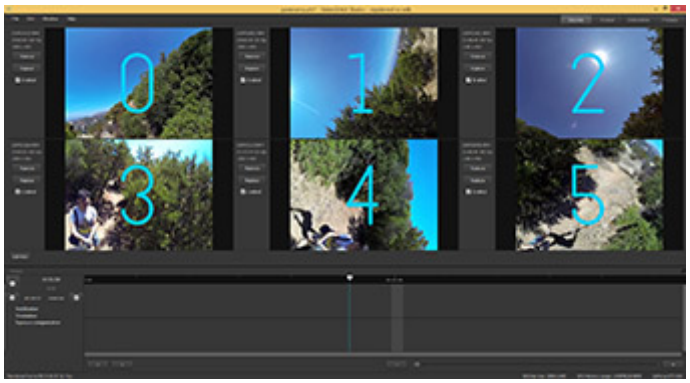

FIGURE 1.8 – Les 6 fichiers vidéos source numérotés importés dans Studio

Synchronisation des vidéos

Lors de la capture, les caméras sont bien souvent démarrées en décalées. Il s'agit donc de resynchroniser les vidéos enregistrées.
Pour cela, la méthode la plus pratique est le *motion*, où le logiciel analyse les mouvements de chaque vidéo : en effet les caméras étant fixées sur la même monture, elles ont donc décris les mêmes mouvements dans l'espace, mais en décalé sur les vidéos.

6. Aussi appelée *rig*

Calibration

À cette étape est reconstituée la sphère ; ce qui consiste à retrouver la position relative de chaque caméra par rapport à la monture.
À un même instant des vidéos, choisis par l'utilisateur, le logiciel trouve des points de contrôle entre les images des différentes caméras, c'est-à-dire des points identiques de la scène présents sur au moins deux images. C'est ici que les zones de recouvrements entre les images trouvent leur importance.
Une fois les points trouvés, les vidéos peuvent être superposées et être assemblées [7].

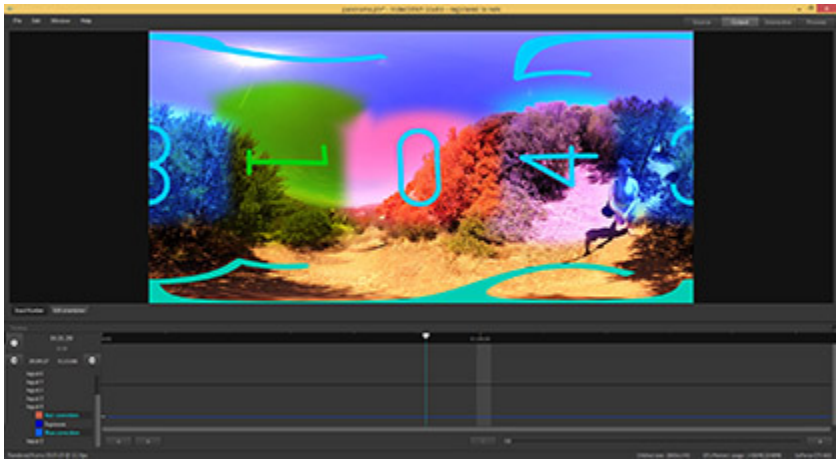

FIGURE 1.9 – Équirectangulaire avec angle de capture colorisés et numéros des 6 sources

Correction d'exposition

Chaque caméra ayant filmé dans une direction, les captures vidéos auront chacune des expositions différentes. De même, les balances des blancs peuvent différer entre les vidéos. Le logiciel corrige localement ces deux paramètres pour homogénéiser le panorama final.

Stabilisation

La stabilisation consiste à lisser les micros mouvements des vidéos, qui ont pu se produire lors de la capture, par exemple si la monture s'est déplacée dans l'espace lors de l'enregistrement. Cette étape est nécessaire : regarder dans un casque de réalité virtuelle une vidéo non stabilisée est très désagréable.

Correction de l'orientation

Il s'agit simplement ici de redonner sens aux notions de haut et de bas dans la vidéo 360. En effet, si la calibration a permis de retrouver les positions des caméras par rapport à la monture, le logiciel n'a aucune information sur la position de la monture par rapport

7. C'est ce qu'on appelle le *stitch*

à la scène. Comme le montre la figure 1.10, la correction s'effectue par un clic de souris enfoncé et glissé jusqu'à déplacer l'équirectangulaire dans la position voulue. Un bon résultat est obtenu quand l'horizon est redressé.

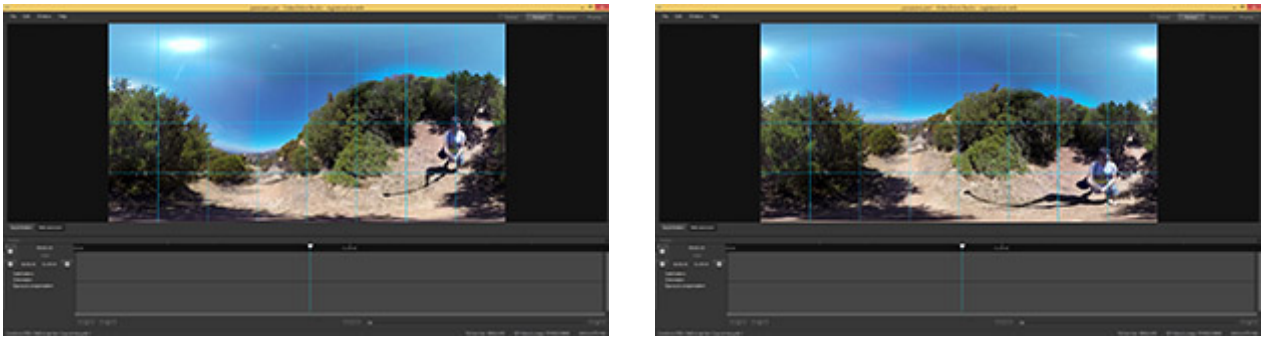

FIGURE 1.10 – Correction de l'orientation : l'horizon est redressé

Exportation

Enfin, la vidéo 360 peut être exportée sous la forme d'un équirectangulaire, avec possibilité d'intégrer le son d'une des caméras.

FIGURE 1.11 – Équirectangulaire final

1.3.2 Vahana VR

Ce nouveau logiciel, dont la sortie stable devrait s'effectuer en février 2015 uniquement sous Windows, permet de réaliser le travail de Studio en temps réel, ainsi qu'une diffusion de l'export en *streaming live*. Les vidéos sont ici directement capturées des caméras via des cartes d'acquisitions, et après une étape de calibration, sont assemblées automatiquement. Le logiciel assure lui-même la bonne synchronisation des vidéos.
Le panorama peut être alors exporté sur différents flux en même temps : streaming RTMP, via une carte d'acquisition pour une sortie HDMI ou SDI, sur le disque dur. Ce logiciel ouvre la possibilité du *streaming* 360 à des événements *live*, comme les concerts, le sport, les défilés ou encore les conférences.

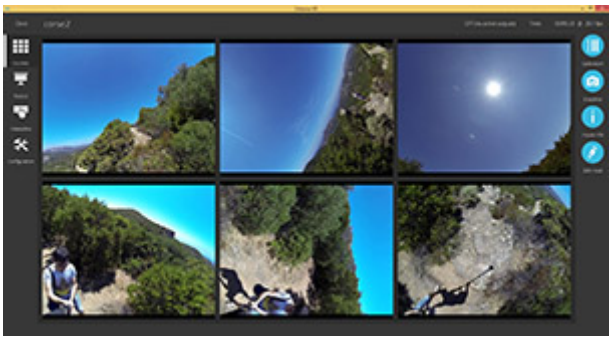

FIGURE 1.12 – 6 entrées vidéos sur Vahana VR

FIGURE 1.13 – Équirectangulaire assemblé après calibration des vidéos

1.3.3 VideoStitch Player

C'est un lecteur de vidéo 360, développé par Alexis Pontin lors de son TN10 au précédent semestre de Printemps 2014, sous Windows, Mac OS X et Linux. Le logiciel permet de lire des vidéos 360 depuis un fichier vidéo équirectangulaire ou depuis un flux RTMP de Vahana VR. La vidéo peut-être lue dans un mode interactif avec utilisation de la souris pour déplacer le regard, ou en mode Oculus Rift, où elle est projetée dans le casque de réalité virtuelle. L'utilisation du casque est tout à fait naturelle : le casque disposant d'un gyroscope et d'un accéléromètre, il renvoie au lecteur la position de la tête de l'utilisateur. Regarder une vidéo avec l'Oculus Rift depuis un flux RTMP de Vahana VR est une expérience toute à fait singulière : le regard est téléporté à la place de la caméra. Il est possible de se voir soi-même en direct, créant un dédoublement du corps et du regard [8].

FIGURE 1.14 – Vue interactive dans le Player, de l'équirectangulaire exporté

FIGURE 1.15 – Vue Oculus dans le Player, de l'équirectangulaire exporté

1.3.4 VideoStitch SDK

8. Une démonstration est disponible en ligne : https://www.youtube.com/watch?v=fQGLqAg0eRU

FIGURE 1.16 – CUDA

Ces applications s'appuient toute trois sur une librairie développée séparément, en C++/CUDA. C'est cette librairie qui implémente et exécute tous les algorithmes de synchronisation, calibration, gestions des flux vidéos et audio et corrections vidéos. Toutes ces opérations sont très coûteuses en temps de calcul. C'est pourquoi la techologie CUDA de nVidia est utilisée pour exploiter le potentiel de la carte graphique [9]. En effet un *GPU* est composé de milliers de cœurs fonctionnant en parallèles. Or la librairie travaille sur des images vidéos, c'est-à-dire des matrices de pixels, avec des calculs massivement parallélisables[15].

Cette librairie est distribuée sous forme de *SDK* [10], sous Windows, Mac OS X et Linux, pour des clients souhaitant concevoir leur propre caméra ou un logiciel exploitant les opérations *stitch*.

1.3.5 Stratégies

Si VideoStitch est pendant un certain temps restée une petite start-up développant un outil de post-production pour vidéos 360 [11], l'entreprise ambitionne désormais d'être le leader du marché.
Le marché est encore jeune, mais se décidera certainement en 2015 et s'alignera probablement sur un leader. La technologie sera suffisamment mature, et des rachats par de grands groupe sont probables, comme cela s'est produit dans le marché de la réalité virtuelle en 2014, avec le rachat de l'Oculus Rift par Facebook [10] par exemple.
Ce marché permettra également l'émergence de nouveaux contenus 360 pour les appareils de réalité virtuelle, comme du cinéma ou de la télévision 360. VideoStitch Studio et Vahana VR sont déjà des outils pertinents pour créer ces nouveaux types de contenus, en particulier Vahana VR pour le streaming 360 en direct. Pour cela il est important que Vahana VR soit le premier produit sur le marché de la 360 *live* et qu'il en prenne la tête : c'est l'opportunité pour VideoStitch de « décoller ».

Cependant d'autres pistes seront ensuite à développer. En effet, avec ces quatre produits l'entreprise se destinerait seulement à une position de service par rapport à des entreprises développant des services 360 ou des vidéos 360. Le développement d'une solution complète incluant une caméra permettrait de dépasser cette position et atteindre le segment du grand public, incluant les entreprises non spécialisées dans la production vidéo ou la réalité virtuelle.

9. Ou *GPU* pour *Graphics Processing Unit*
10. *Software Development Kit*
11. VideoStitch Studio (section 1.3.1 p.10)

2. Présentation du stage

2.1 Le sujet

Le sujet du stage était initialement « *Developer C++/Qt. work on a challenging and immersive 360 video software* ».
Bien que le sujet soit assez général, le domaine de l'entreprise se trouvait être très intéressant et durant l'entretien qui s'est déroulé à la mi-mai, quelques sujets ont été abordés : Nicolas Burtey a proposé de travailler sur le VideoStitch Player d'une part, à la suite d'Alexis Pontin et pour terminer son travail, et d'autre part de participer au développement du nouveau produit, Vahana VR, qui allait débuter dans quelques semaines.
M. Burtey n'était pas encore certain de développer Vahana VR, et envisageait également à la conception du *stitch* stéréoscopique 3D [1] : il proposa également un sujet sur la prise en charge de la 360 stéréo dans Videostitch Studio.
Le sujet final serait décidé la première semaine du stage.

Le stage fut finalement précédé par un CDD, les missions retenues se sont alors simplement étendues du début du CDD à la fin du stage.
Le sujet réel a donc été arrêté à deux missions :

1. Simplifier et améliorer le *workflow* de développement, d'une gestion mono-logicielle vers une gestion multi-logicielle, en y intégrant le VideoStitch Player et Vahana VR.
2. Développer et le déployer un système de *plugins* basés sur des caméras et des cartes d'acquisitions, et augmentant les capacités entrées/sorties de Vahana VR.

Ces deux missions [2] trouvent comme sujet commun *l'amélioration et la conception de workflows à l'usage de l'équipe de développement*. La première consiste en ce sens à l'amélioration du système de développement et des outils qui y sont associés ainsi qu'à l'intégration des logiciels de l'entreprise. La seconde permet, elle, par un système indépendant et modulaire, d'apporter nouvelles capacités à Vahana VR, qui seront utilisées par les ingénieurs pour le développement de ce logiciel.

1. Qui consiste, brièvement, en la conception de deux équirectangulaires pour la même scène filmée : un pour chaque œil qui, en exploitant l'effet de la parallaxe, recrée un effet de relief[16] [17]
2. La 3D s'est révélée être une tâche trop ardue, après un rapide état de l'art du sujet par les ingénieurs : même si les articles de recherche et brevets sont nombreux, aucune application industrielle n'a encore aujourd'hui aboutis.

2.2 Les méthodes de travail

Le travail de l'équipe des développeurs s'appuie en grande partie sur la méthode *Scrum*, et sur quelques pratiques de qualité logicielle issues de l'*Extreme Programming*, comme l'intégration continue. Ces deux méthodes sont des *méthodologies agiles*[18].

Scrum présente un certain nombres de pratiques ; les éléments utilisés dans le cadre de VideoStitch sont essentiellement des événements et des artefacts décrits dans le *Scrum guide*[19] :

- **Sprint** : le principe de la méthode repose sur le découpage du projet en périodes de temps, nommées *sprint*, dont la durée à VideoStitch fut de une ou deux semaines selon les besoins. Chaque sprint possède un but à accomplir durant la période allouée, ce qui permet une grande réactivité sur les besoins du projet, qui est, dans le cas de VideoStitch, l'entreprise elle-même.
- **Scrum meeting** : réunion qui marque la fin d'un sprint et permet la préparation du suivant. Elle se déroule en trois temps :
  1. *Revue du sprint* : c'est un passage en revue de ce qui a été réalisé et de ce qui ne l'a pas été durant le sprint, ainsi qu'une démonstration de ce qui a été complété. Cela permet de valider l'objectif et d'ajuster la planification du projet en fonction de son avancement.
  2. *Retrospective du sprint* : elle a pour but une amélioration continue de l'environnement de travail et des méthodes utilisées. Chacun peut proposer des éléments d'amélioration, qui après vote, seront mis en place dès le sprint suivant.
  3. *Planification du sprint suivant* : l'équipe détermine le but du sprint suivant, et à partir du *backlog* précise son contenu concret en liste de tâches.
- **Backlog** : c'est une liste de l'ensemble des tâches (appelées *tickets*) de tout ce qui est nécessaire pour enrichir le projet. Elle initialement crée avant le début du projet, et enrichie lors des sprints en fonction de l'apparition des besoins nouveaux.
- **Status des tickets** : après être choisis du *backlog* au sprint courant, un ticket doit passer trois statuts pour être considéré comme réalisé :
  1. *Selected for development* : le ticket n'est pas réalisé.
  2. *In progress* : le ticket est en cours de réalisation par un développeur.
  3. *QA* : le ticket est réalisé et attends une validation d'une autre personne.
  4. *Done* : le ticket est réalisé et validé.
- **Stand-up** : ou encore *daily scrum*, est une courte réunion de 15 minutes se déroulant chaque matin pour faire le point entre tous les membres de l'équipe. Chacun aborde ce qu'il a effectué la veille, les difficultés éventuelles qu'il a rencontré et ce qu'il compte réaliser aujourd'hui pour atteindre l'objet du sprint.

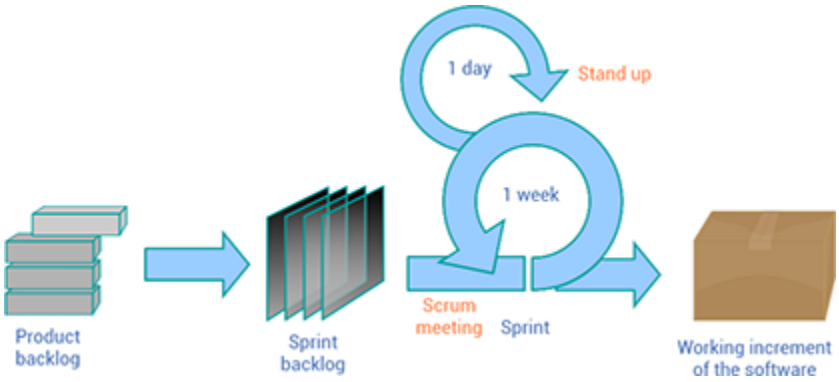

FIGURE 2.1 – Illustration de la méthode Scrum[20]

Cependant, le VideoStitch SDK et les ingénieurs y travaillant ne participent pas aux sprints : leur travail étant plutôt organisés sous formes de missions individuelles, en fonction des besoins des applications de l'entreprise. Ils participent toutefois aux *stand-up* et au *retrospectives*.
Le présent stage a suivi l'équipe dans ses sprints, en l'assistant parfois sur quelques tickets, tout en maintenant en priorité les deux missions de fond présentées plus haut, dans la section Le sujet (section 2.1 p.15).

En outre, un wiki interne est utilisé pour documenter et mutualiser les différentes connaissances et apprentissages de l'équipe.
Enfin, toutes les communications écrites et orales se font en anglais, du fait de la présence dans l'équipe de personnes de langue natale espagnole, allemande ou flamande.

2.3 Les outils utilisés

Les quatre produits de VideoStitch sont codés en *C++*, avec utilisation des dernières possibilités de C++11 ; VideoStitch SDK utilise également la bibliothèque *CUDA* pour les calculs intensifs et parallélisables. Le compilateur natif de chaque OS est utilisé : g++ pour Linux, clang pour Max OS X et cl sous Windows.

FIGURE 2.2 – Logo de Qt

Les trois applications sont développées avec le framework C++ *Qt*, qui permet d'assurer la portabilité sur les plateformes Windows, Linux et Mac OS X du même code source. En outre, Qt propose un certain nombre de classes, en addition de la STL de C++, facilitant le développement d'applications de haut niveau. Enfin ce framework est connu pour son système de signals/slots ouvrant la voie à la programmation événementielle, et asynchrone : en addition à l'appel de fonctions, telle qu'existant depuis le C, un objet peut émettre des *signaux*. D'autres objets peuvent recevoir à leur tour ces

signaux pour exécuter des fonctions, appelées dans ce cas *slot*.[21]

Tous les codes source sont partagés et versionnés grâce au système de gestion de version distribué *Git*[22]. Ce logiciel permet à l'équipe de travailler avec des *dépôts* : un dépôt Git est simplement un dossier associé à une base de donnée archivant l'historique de toutes les modifications qui y ont été effectuées. Chaque développeur possède une copie de ce dépôt et de l'historique associé sur son ordinateur. Les développeurs se transmettent régulièrement les modifications qu'ils ont apportées pour synchroniser les historiques de toutes les copies : ainsi, chaque membre de l'équipe sait qui a modifié quoi et quand.

FIGURE 2.3 – Logo de Git

En pratique, comme le montre la figure 2.5, un serveur est utilisé pour sauvegarder les modifications faites en servant de pont entre les développeurs : chacun y propose ses modifications et l'interroge sur celles des autres.

FIGURE 2.4 – Logo de Buildbot

Enfin, à chaque modification apportée sur un dépôt Git, un serveur *Buildbot* va exécuter une routine, comme le montre la figure 2.6 : le produit concerné par la modification sera compilé, subira un ensemble de tests de qualité et son installateur sera généré et mis à disposition sur une page de téléchargement à usage interne.
Ces installateurs pourront être utilisés par n'importe quelle personne de l'entreprise pour des tests, en compléments avec les tests automatiques de qualité, pratique d'*intégration continue*, qui permettent de vérifier à chaque modification effectuée qu'aucun bug, ni régression n'a été introduit[23].

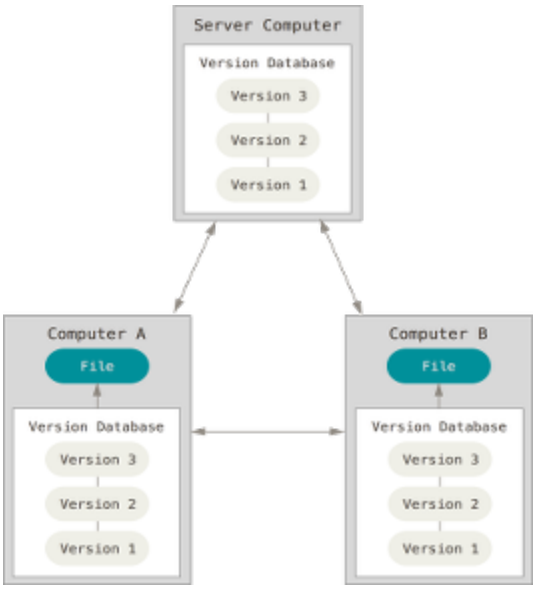

FIGURE 2.5 – Schéma de gestion de version distribué[24]

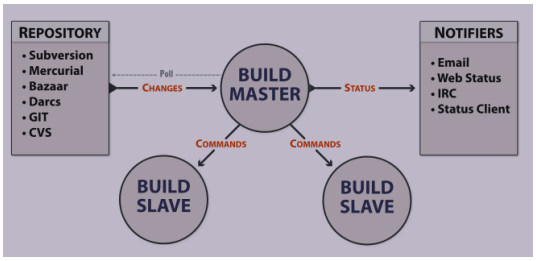

FIGURE 2.6 – Schéma de fonctionnement de Buildbot[25]

3. Amélioration du workflow de développement

3.1 Définition de la mission

3.1.1 Contexte

La première tâche de ce stage fut la prise de contact avec les méthodes de travail, les outils utilisés, les logiciels développés, et leurs *workflows* de développement.
Ce qui est désigné par *workflow* est un « flux de travaux, [c'est-à-dire] une suite de tâches et d'opérations effectuées par [...] un groupe de personnes »[26], ici l'équipe de développement de VideoStitch. Un *workflow* est une représentation des différentes tâches à accomplir pour installer un poste de travail vierge, et compiler une des applications. Ce n'est pas l'architecture des logiciels, ni leur développement en lui-même, mais comment, à partir des sources et des dépendances, il est possible d'arriver au produit compilé et distribuable au client.
Ce processus met en œuvre les concepts présentés à la section Les outils utilisés (section 2.3 p.17), s'articulant autour du développement logiciel proprement dit[27][28] :

- La gestion de versions, assurée ici avec Git[22], qui permet de récupérer les codes sources.
- La chaîne de compilation, dépendante de l'OS cible et automatisée avec Buildbot [29][23].
- Les tests d'intégration[23] réalisés par Buildbot.
- La création des installateurs et leur mise à disposition sur une page de téléchargement interne.

Cette mission porte essentiellement sur l'amélioration de la chaîne de compilation des applications et la création des installateurs.

3.1.2 Le *workflow* de développement de VideoStitch Studio

Il s'agissait tout d'abord de comprendre comment était organisé le *workflow* de VideoStitch Studio. Les codes source et les dépendances avaient été réparties en quatre dépôts Git :

- VideoStitch-sdk, contenant les codes source de VideoStitch SDK

— VideoStitch-base, contenant des code source partagés avec le dépôt du VideoStitch Player
— VideoStitch-apps, contenant les codes source de VideoStitch Studio
— VideoStitch-deps [1], contenant les dépendances Windows de VideoStitch Studio nécessaires à sa compilation

Sous les systèmes Linux et OS X, il a été décidé d'utiliser les gestionnaires de paquets apt et Macport pour installer les dépendances logicielles de Studio. Pour Windows, le dépôt VideoStitch-deps tenait les dépendances sous des versions stables et testées.

Compiler et créer l'installateur de Studio depuis un poste de travail vierge requérait alors certaines étapes, représentées sous forme d'un diagramme de séquence sur la figure 3.1. Ces étapes étaient essentiellement des appels à des commandes et à des scripts internes (écrits en bash pour Linux / Mac et en batch pour Windows) pour faire le lien entre les dépôts :

1. Cloner les dépôts VideoStitch-deps, VideoStitch-base et VideoStitch-apps.
2. Copier, via un script présent sur VideoStitch-base, les codes source partagés.
3. Copier, via un script batch présent sur VideoStitch-deps, les dépendances pour Windows.
4. Récupérer la dernière version du VideoStitch SDK compilée par Buildbot, via un script présent sur VideoStitch-apps.
5. Générer le Makefile [2] avec le logiciel qmake, fournis avec Qt.
6. Compiler avec le logiciel jom [3], fournis avec Qt.
7. Générer l'installateur avec un script présent sur VideoStitch-apps, si l'on désirait distribuer le logiciel compilé.

3.1.3 Problématique

Représenter ce flux présente l'intérêt de pouvoir ensuite le simplifier et d'automatiser un maximum des processus requis. Et cela était devenu nécessaire, pour deux raisons :

— L'entreprise commençait sa croissance, impliquant de former les nouveaux arrivants à un *workflow* de développement devenue complexe à mesure du temps. Il fallait en simplifier le fonctionnement maintenant que Studio et son développement était devenus matures [5], ce qui permettrait de gagner en efficacité sur le développement proprement dit.

1. Présentées plus en détails dans Intégration des dépendances du VideoStitch Player (section 3.2.1 p.22)
2. Fichier contenant un ensemble de règles destinées au compilateur et à l'éditeur de liens.
3. Ce logiciel fait appel au compilateur du système ainsi qu'à l'éditeur de lien pour générer le logiciel selon les règles décrites dans le Makefile généré.
4. Écrit en anglais pour des raisons de cohérence entre les commandes utilisés.
5. La première version était sortie, et la bêta de la seconde version était en cours

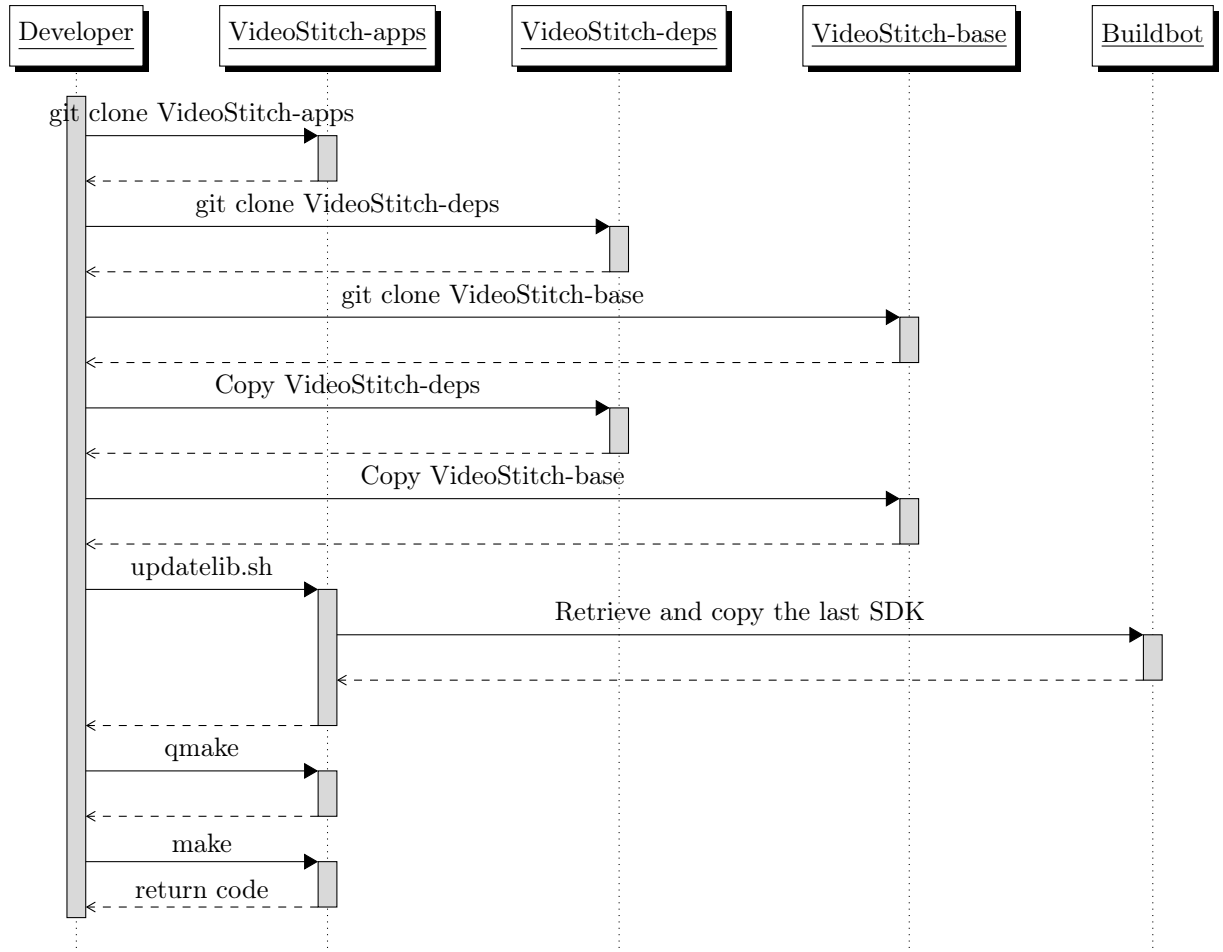

FIGURE 3.1 – Diagramme de séquence du *workflow* de développement de Studio (juillet 2014) [4]

— Vahana VR et le Player avaient débuté leurs développements ; pourtant ces trois produits, avec Studio, ont en commun l'utilisation de VideoStitch SDK : les logiciels étant très proches, il était nécessaire d'unifier leurs *worklows* de développement.

3.1.4 Objectifs

Les objectifs retenus ont été :

— Faire évoluer le *workflow* de développement vers une gestion multi-logicielle sur le dépôt VideoStitch-apps et en simplifier le fonctionnement.

— Intégrer VideoStitch Player et Vahana VR à ce *workflow* de développement.

— A cela s'est ajouté, avec le départ d'Alexis, un suivi du Player pour en assurer sa maintenance.

3.2 Réalisation [6]

3.2.1 Intégration des dépendances du VideoStitch Player

Un programme dépend, pour sa compilation et son exécution, d'autres *paquets* logiciels [30]. Ce sont des archives contenant des *bibliothèques logicielles*, c'est-à-dire des ensembles de fonctions déjà compilées et utilisables par le programme.
Une bibliothèque logicielle sous Windows se constitue de trois types de fichier[31] :

— Les *headers* (fichiers .h), qui déclarent les prototypes des fonctions utilisables et accessibles de la bibliothèque.
— Les bibliothèques (fichiers .lib et .dll), qui contiennent le code des fonctions. Les fichiers .lib seront copiés dans le programme, alors que les fichiers .dll seront chargés lors du démarrage du programme[31].

À cela s'ajoute des spécificités autre à celle du système d'exploitation : un programme, et donc ses dépendances, est spécifique à la plateforme visée (32 bits ou 64 bits)[32] mais aussi à la configuration compilée (*debug* ou *release*)[33].
La configuration *debug* permet aux développeurs de générer une version pour le déboguage, quand la configuration *release* est utilisée comme une version finale et optimisée destinée à l'usage du programme.
Enfin, le choix de la plateforme dépend du processeur et du système d'exploitation du client. Les produits de VideoStitch sont distribués seulement en 64 bits : les machines pour les faire fonctionner demandent du matériel récent donc compatible 64 bits. De plus cela réduit la maintenance à une seule plateforme.

Le dépôt VideoStitch-deps était déjà logiquement organisé sous cette même forme, présentée par le listing 1. Cette organisation convenait aux nouveaux besoins et a été gardée ainsi.

```
/
├── src ..... Archives des versions des dépendances utilisées
└── x64 ..... La plateforme 64 bits Windows
    ├── bin ..... Les bibliothèques .dll
    │   ├── debug ..... Les .dll spécifiques debug
    │   └── release
    ├── include ..... Les fichiers headers .h
    └── lib ..... Les bibliothèques .lib
```

Listing 1 – Dépôt VideoStitch-deps

6. Nicolas Lopez, Julien Fond, Wieland Morgenstern et Jean Duthon m'ont beaucoup aidé et, par leurs corrections, ont largement contribué aux résultats présentés ici.

Une séparation des dépendances par logiciel et par version aurait été possible, mais complexifiant inutilement la structure. De nombreuses dépendances sont communes entre les trois logiciels et sont utilisées dans les même versions.

Ainsi, dans le cadre du Player, deux dépendances ont été ajoutées :

- **libVLC** : SDK de VLC, cette bibliothèque fournit des capacités de lecture multimédia.[34]
- **Oculus SDK** : cette bibliothèque permet à une application d'exploiter le casque de réalité virtuelle Oculus Rift.[35]

Les dépendances Qt et crashrpt [7] étaient déjà présentes et utilisées par Studio.
En parallèle, les dépendances libVLC et Oculus SDK ont été installées sur les Buidlbots Mac et Linux.
Quant aux dépendances requises par Vahana VR, elles étaient les même que celles utilisées Studio, donc déjà présentes dans le dépôt.

3.2.2 Intégration au *workflow* du VideoStitch Player et de Vahana VR

Il fallait ensuite intégrer le Player et Vahana VR à VideoStitch-apps.
La première étape a été de rapatrier le dépôt VideoStitch-base et de l'inclure avec son historique dans le dépôt VideoStitch-apps. L'annexe A présente plus en détail l'opération permettant, grâce à Git, de déplacer un ensemble de fichiers d'un dépôt à un autre tout en préservant l'historique des modifications du dépôt importé.
De la même manière, furent importés les codes source de Player et Vahana VR de leurs dépôts respectifs.
L'opération fut relativement aisée, car le dépôt VideoStitch-apps présentait déjà une organisation regroupant plusieurs programmes, mais centrés sur le seul logiciel VideoStitch Studio [8]. L'architecture ne nécessitait alors que quelques modifications, et a été fixée à la hiérarchie présentée par le listing 2.

L'architecture convenant, les fichiers projets Qt nécessitaient d'être refactorisés, c'est-à-dire les réécrire pour les adapter aux nouveaux besoins.
Les applications Qt définissent des projets (extension .pro) décrivant les fichiers source, les ressources et les dépendances utilisées par le logiciel, ainsi que les paramètres pour le compilateur et l'éditeur de liens (*linker*). Ce sont à partir de ces fichiers que le programme qmake construit les instructions de la chaîne de compilation qui seront exécutés par jom[21]. C'est ce qui permet la portabilité des projets Qt : un seul projet est écrit, et l'application peut être compilée sur plusieurs systèmes.

7. Envoi de rapports automatiques lors d'un *crash* du logiciel https://code.google.com/p/crashrpt/
8. On distingue logiciel et programme, dans le sens où un logiciel est composé de un ou plusieurs programmes, c'est-à-dire des fichiers binaires, ainsi que des fichiers de configurations, images, etc.[36]

```
/
├── bin ..... Dossier de compilation et dll copiées de VideoStitch-deps
│   └── x64
│       ├── debug
│       └── release
├── installer ..... Scripts de conception des installateurs
├── external_deps ..... Headers et lib copiés de VideoStitch-deps (Windows)
│   ├── include
│   └── lib
├── packager ..... Scripts de conception des installateurs
└── src ..... Codes source des applications
    ├── libvideostitch ..... Contient les headers copiés du VideoStitch SDK
    ├── libvideostitch-base .. Programme commun au Player, Studio et Vahana VR
    ├── videostitch-player-gui ..... VideoStitch Player
    ├── libvideostitch-gui .. Programme commun à Studio et Vahana VR seulement
    ├── videostitch-studio-gui ..... VideoStitch Studio
    ├── batchstitcher. ..... Programme inclus avec Studio
    └── vahana-vr ..... Vahana VR
```

Listing 2 – Dépôt VideoStitch-apps

Dans ce cas ci, les trois applications étant proches, il était intéressant de partager la configuration des différents projets. Les codes source communs avaient déjà été regroupés dans les programmes libvideostitch-base et libvideostitch-gui.

Enfin, quelques scripts ont été réécrits. Comme vu avec la section Le *workflow* de développement de VideoStitch Studio (section 3.1.2 p.19), le développeur doit récupérer la bibliothèque du VideoStitch SDK et, sur Windows, les dépendances. Ces ressources se trouvent sur des dépôts séparés, c'est pourquoi il était nécessaire de pouvoir les récupérer facilement. Git étant l'outil le plus utilisé et son utilisation se réalisant en ligne de commande, des scripts ont été écrits pour être utilisables comme des commandes additionnelles à l'usage de Git.
De plus tous les scripts batch (pour Windows) ont été supprimés et fusionnés avec les scripts bash. Si bash est supporté sur tous les Linux et les Mac OS X, l'utilisation systématique de Cygwin sur Windows par l'équipe a assuré le support sur cette plateforme également.
Son utilisation est très simple : le listing 3 présente une copie de l'aide affichée.
Dans le cas de `lib`, le script va détecter le système de l'utilisateur, puis télécharger avec le programme curl la dernière version compilée du VideoStitch SDK pour le système et la configuration voulue, pour copier les *headers* et les bibliothèques dans les dossiers attendus sur le dépôt VideoStitch-apps. Le script automatise simplement une action de recherche d'une version compilée par Buildbot pour en extraire et copier les fichiers.

```
Usage: ./updatelib.sh <repository> [configuration]
Update the actual folder with the required <repository>.

<repository> is mandatory and other parameters are optionals.
Parameters can takes the following values:
<repository>      'deps', 'lib' or 'all' (default)
[configuration]   'release' (default) or 'debug'
```

Listing 3 – Aide affichée par la commande ./update.sh

Dans le cas de `deps`, le script va créer une copie locale de VideoStitch-deps, puis en copier les headers et les bibliothèques dans les dossiers de VideoStitch-apps.

La figure 3.2 montre la nouvelle utilisation du workflow désormais disponible.

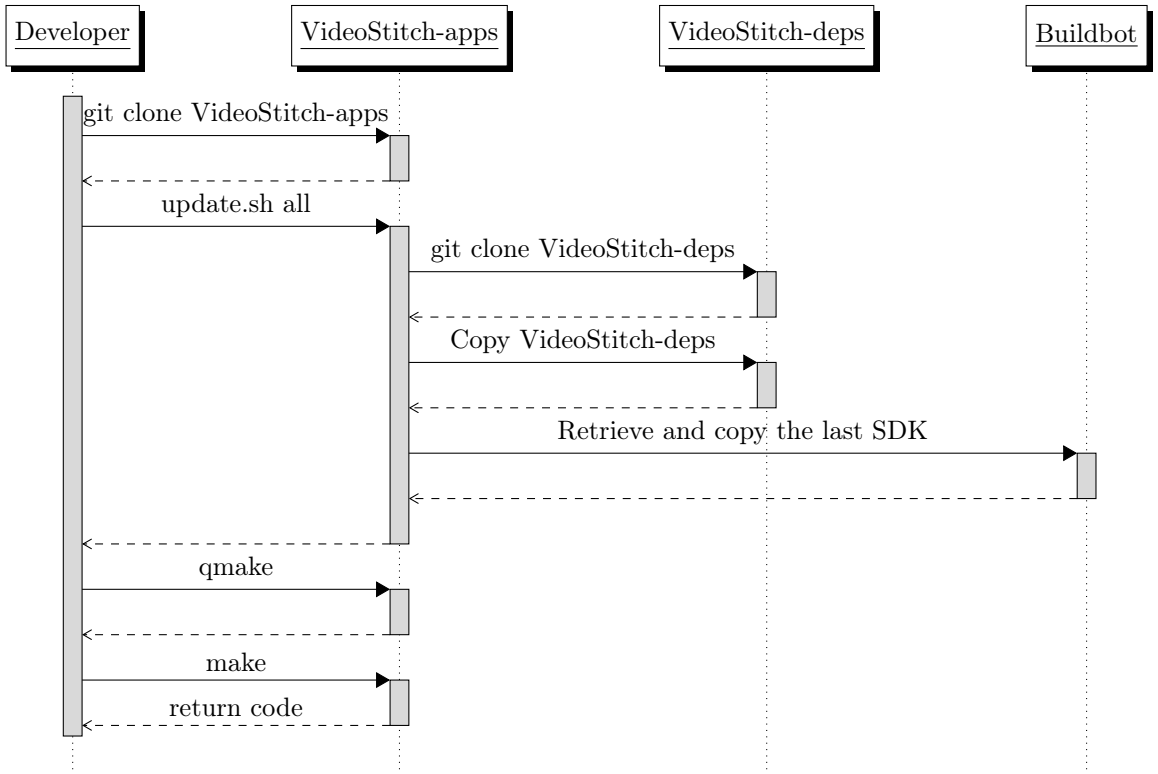

FIGURE 3.2 – Diagramme de séquence du *workflow* de développement des applications de VideoStitch (Player, Studio, Vahana VR) (septembre 2014)

3.2.3 Génération des installateurs

Si la compilation des trois produits est désormais possible, il s'agit désormais de pouvoir distribuer au client une version finale et utilisable.
Le logiciel doit être fournis en version *release* et optimisée, avec ses dépendances associées.

Cependant la compilation produit un certain nombre de fichiers supplémentaires au programme compilé, essentiellement pour assister le post-développement du logiciel. L'idée est le même sur les trois OS : il s'agit donc de construire un paquet contenant seulement les fichiers nécessaires, le logiciel et ses dépendances.

Le logiciel InnoSetup a été utilisé sur Windows pour générer les installateurs des trois applications. Comme pour les projets Qt, les codes des installateurs ont été un maximum mises en commun. Le listing 4 présente un exemple d'une configuration d'installateur, fichier qu'il suffit ensuite de passer en argument en ligne de commande au programme Inno Setup pour générer l'installateur.

```
1  [Setup]
2  AppName="VideoStitch Player"
3  #include "common-setup.iss"
4
5  [Dirs] # will be created and then deleted at uninstall
6  Name: "{app}"; Flags: uninuninsalwaysuninstall
7
8  [File] # only necessary files from bin\ will be copied
9  Source: "bin\x64\release\videostitch-player.exe"; Destdir: "{app}";
10 Source: "bin\x64\release\libvideositch-base.exe"; Destdir: "{app}";
11 Source: "bin\x64\release\libvlc.dll"; Destdir: "{app}";
12 Source: "bin\x64\release\Qt5*.dll"; Destdir: "{app}";
13
14 [Run]
15 Filename: "{app}\vcredistx_64_2013.exe"; Parameters: "/q /norestart"; WorkingDir:
   ↪  "{app}";
16 StatusMsg: "Installing Microsoft Visual 2013 (x64) Redistributable Package"
```

Listing 4 – Extrait de la configuration de l'installateur de VideoStitch Player

De même, les paquet Linux générés sont très simples, comme le présente le listing 5.

```
/
├─ bin
│  └─ videostitch-player ..... Le programme
└─ lib
   ├─ libvideostitch-base.so
   ├─ libvlc.so
   ├─ libQt5Core.so
   └─ libQt5Gui.so
└─ launcher ..... Lance le programme après initialisation des librairies
```

Listing 5 – Paquet Linux de VideoStitch Player

3.2.4 Automatisation de la chaîne de compilation

Il s'agit désormais d'écrire les routines pour permettre à Buildbot de générer les quatre installateur des applications de VideoStitch.
Deux concepts sont importants dans Buildbot : les *builders* et les *schedulers*. Les premiers sont la suite des étapes à accomplir pour générer les logiciels, quand les seconds sont ceux qui vont les déclencher après une modification sur un dépôt, ou après la fin d'un *builder* : par exemple, après la compilation de VideoStitch SDK, les applications sont générées pour tester le bon fonctionnement du SDK.
Le système de *schedulers* et de *builders* étant déjà créé pour générer Studio sur Windows, Mac et Linux, il suffisait de rajouter les différentes étapes pour générer le Player et Vahana VR. Ces étapes sont écrites en Python dans un fichier de configuration.

Pour exemple, le listing 6 présente un extrait de l'utilisation du *builder* du dépôt VideoStitch-apps pour Windows, qui a été précédemment définis avec l'identifiant `windows_apps` :

```
1  windows_apps = BuildFactory() # Initiate a new build
2  windows_apps.addStep(getGitRepo("apps", "git clone VideoStitch-apps"))
3  windows_apps.addStep(ShellCommand(command=["bash", "update.sh", "all", "release"],
   ↪  workdir="VideoStitch-apps"))
4  windows_apps.addStep(ShellCommand(command=["qmake", "CONFIG+=release"],
   ↪  workdir="VideoStitch-apps/src/vahana-vr")
5  windows_apps.addStep(ShellCommand(command=["make"],
   ↪  workdir="VideoStitch-apps/src/vahana-vr"))
```

Listing 6 – Extrait du *builder* Windows sur le Buildbot

3.2.5 Maintenance du VideoStitch Player

Enfin, quelques apports mineurs et rapides ont été apportés, pour améliorer l'usage du Player lors des présentations dans les salons et conférences.

Support de l'Oculus Rift DK2

Le Player propose deux modes de vues, comme présenté dans la section VideoStitch Player (section 1.3.3 p.13) : une vue interactive, avec un déplacement dans la sphère 360 avec la souris et un affichage sur un écran standard, et une seconde vue utilisant le casque de réalité virtuelle Oculus Rift pour le déplacement et l'affichage, créant une vue totalement immersive.
En juillet 2014[37], est sortie la nouvelle version dite DK2 de ce casque encore en développement. Le Player se devait de supporter cette nouvelle version.
Dans un premier temps, après réception du casque, les drivers Oculus ont été mis à jour

et le bon fonctionnement du casque a été testé. Une nouvelle version de l'Oculus SDK avait été mise à disposition, VideoStitch-deps a donc été mis à jour avec les nouvelles bibliothèques et *headers* tout comme les machines Buildbot Mac et Linux.

Une classe `OculusRift` avait déjà été définie pour permettre l'utilisation de l'Oculus SDK. Après la lecture de la documentation de ce SDK et celle du code, le Player a été mis à jour pour utiliser l'API de cette nouvelle version, assurant ainsi le support du DK1 et du DK2.

Passer le *Health and Safety Warning* de l'Oculus

Le nouvel Oculus SDK a vu l'apparition de nouvelles clauses légales et d'un écran d'avertissement appelé *Health and Safety Warning* apparaissant dans le casque lors du démarrage d'une vidéo[35].
Il s'agissait alors d'ajouter une possibilité de passer cet écran, en pressant une manuellement une touche du clavier. Cela ne pouvait être fait automatiquement afin de respecter les clauses.
L'implémentation se réalisa avec la création d'une fonction pour désactiver le HSW (listing 7) et avec une capture d'une touche du clavier pour faire appel à cette fonction (listing 8) via le système signal/slot de Qt.
La classe `OculusRift` propose une gestion de l'oculus et une utilisation à l'ensemble de l'application du Player, via un certain nombre de fonctions publiques et de slots Qt [9]. La fenêtre principale `MainWindow` de l'application utilise la fonction prédéfinie `keyPressEvent` fournie par Qt qui « écoute » les touches pressées par l'utilisateur dans le Player. Ici, quel que soit la touche, un signal est envoyé pour demander la désactivation du HSW.

```
109  void OculusRift::disableHSW() {
110    ovrHSWDisplayState hsw;
111    ovrHmd_GetHSWDisplayState(hmd, &hsw);
112    if (hsw) {
113      ovrhmd_EnableHSWDisplaySDKRender(hmd, false); // hmd object manages the oculus
114    }
115  }
```

Listing 7 – Extrait du fichier oculusrift.cpp

9. Un slot est une fonction pouvant être appellée de manière asynchrone par un signal provenant de la même classe ou d'une autre.

```
90  connect(this, SIGNAL(disableOculusHSW()), oculusRift, SLOT(disableHSW()));
91
92  void MainWindow::keyPressEvent(QKeyEvent *e) {
93    emit disableOculusHSW(); // send the signal, will be catch by every connected slot
94  }
```

Listing 8 – Extrait du fichier mainwindow.cpp

3.3 Bilan et suite

Les modifications apportées, même si elles ont été parfois mineures, ont cependant permis dans leur globalité :

- De permettre une installation d'un nouvel environnement de développement et de son usage relativement aisé, comme le montre la figure 3.2.
- De compiler depuis un même dépôt, quelque que soit l'OS de la machine, les trois applications de VideoStitch.
- D'automatiser la compilation et la génération des paquets de ces trois applications sur Buidlbot, rendant les dernières versions immédiatement utilisables par n'importe quelle personne de VideoStitch.
- D'apporter les modifications nécessaires à l'usage du Player.

Ce système pourrait encore être amélioré, ou même à être repris de zéro. Cependant, une telle réécriture demanderait nécessairement beaucoup plus de temps pour produire un système certes plus correct au regard des besoins actuels, mais moins mature et donc plus sujet à des cas d'utilisations non prévus ou problématiques. De plus, cela demanderait de changer les habitudes de l'équipe de développement.
Cependant, des cas d'utilisations non prévus sont également apparus lors de cette mission, mais ont rapidement été corrigés par l'équipe. Bien souvent, cela était dû à des différences d'interprétations du bash entre Linux et Mac OS X. Les scripts pourraient être écrits en Python pour éviter ces différences et ces erreurs. Leur écriture et leur maintenance serait en outre facile, Python étant relativement simple dans son approche et ses usages.
Enfin, VideoStitch Player nécessiterait d'être amélioré : la vidéo 4K n'est pas encore correctement lue, et le logiciel gagnerait à être finis lors de quelques sprints de l'équipe.

4. Développement d'un système de *plugins* entrées/sorties

4.1 Définition de la mission

4.1.1 Contexte

Dès le début du développement de Vahana VR, la question des entrées et des sorties du logiciel s'était posée. En effet, Studio étant un logiciel de montage de vidéos 360, ses seules entrées sont des fichiers vidéos, issus des caméras de la monture. De même, la seule sortie est le fichier vidéo 360, comme présenté dans la section Exportation (section 1.3.1 p.12).
Ce qui est ici appelé *entrées* sont les données envoyées au programme, quand les *sorties* sont les données émises par ce programme en retour.
Pour permettre la vidéo 360 en direct, Vahana VR, contrairement à Studio, s'affranchit des fichiers vidéos importés et propose de capturer directement les images des caméras, via des cartes d'acquisitions branchées sur la machine utilisée. De même, Vahana VR peut émettre plusieurs fluxs en sortie permettant, par exemple, le *streaming* RTMP ou la réémission vers une carte d'acquisition SDI ou HDMI, en plus de l'export de la vidéo 360 sur le disque dur.

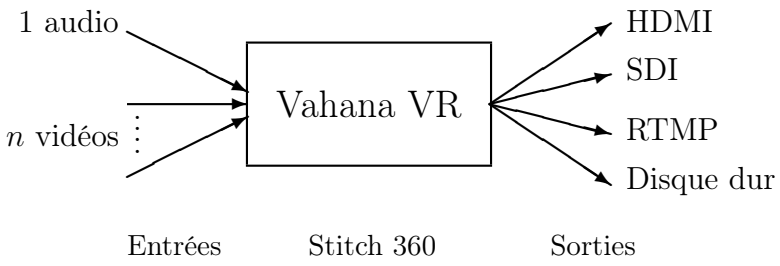

FIGURE 4.1 – Schéma des entrées/sorties de Vahana VR

4.1.2 Problématique

Dès lors, Vahana VR étant destiné à des sociétés de production, il doit être compatible avec un maximum de standards, caméras et cartes d'acquisitions vidéos pour être

facilement adopté. Cependant, tout matériel informatique requérant des pilotes [1] , et, les besoins des clients en entrées/sorties n'étant pas les mêmes, il n'était alors pas possible de concevoir un logiciel monolithique contenant tous les programmes d'entrées/sorties supportés : l'installation de Vahana VR aurait également requis au client une installation de l'ensemble des pilotes.
De plus, un client pourrait souhaiter utiliser du matériel entrée/sortie encore non supporté. Il serait intéressant qu'il puisse réaliser son propre développement dans l'écosystème de Vahana VR ; ainsi le logiciel pourrait devenir virtuellement compatible avec n'importe quel standard, caméra ou carte d'acquisition.
Il fallait donc développer un système assurant à la fois la compatibilité entre Vahana VR et ces matériels, et intégrant les contraintes exposées.

4.1.3 Objectifs

Au début de ce stage, une solution avait déjà été esquissée et était déjà en partie mise à l'œuvre. Cependant un certain travail de développement était encore nécessaire.

Les objectifs retenus pour cette mission ont donc été :

- Développer les entrées HDMI [2] et SDI [3].
- Développer les sorties HDMI et SDI.
- Développer des entrées/sorties Ethernet et PCIe [4] spécifiques, à la demande d'un client.
- Déployer l'ensemble des *plugins* sur un dépôt séparé.
- Documenter et vérifier la bonne intégration des *plugins* dans Vahana VR.

(a) – HDMI (b) – SDI (c) – Ethernet (d) – PCIe

FIGURE 4.2 – Illustration des différents connecteurs utilisés par Vahana VR

1. Programme permettant au système d'exploitation d'interagir avec le matériel couvert par ce pilote[38].
2. *High Definition Multimedia Interface*, norme de diffusion audio/vidéo numérique[39].
3. *Serial Digital Interface*, protocole de diffusion vidéo numérique[40].
4. *PCI Express*, standard de connexion de cartes d'extension sur la carte mère d'un ordinateur[41].

4.2 Réalisation

4.2.1 Architecture de la solution

Il s'agissait tout d'abord de saisir le concept de la Programmation Orientée Composant, et la manière dont il est appliqué pour Vahana VR. Cette approche propose d'utiliser différents *composants*, c'est-à-dire des briques logicielles qui peuvent interagir entre elles. Tout comme les dépendances [5], un composant va fournir un ensemble de fonctions, présentes dans des bibliothèques (fichiers .dll sous Windows). Elles sont déclarées dans les *headers* qui accompagnent ces bibliothèques, afin de pouvoir être utilisées par les autres composants.[42]
Ici, on parle un peu plus spécifiquement de *plugins* et de *programme client*. Les *plugins* peuvent être chargées et utilisées par un programme client, ici Vahana VR. Leurs fonctions sont donc regroupées en des modules externes au logiciel qui peut y faire appel si besoin, ou non. Leurs interfaces sont, cette fois, définies dans le logiciel.[43]
Un *plugin* est donc utilisé indirectement via ses interfaces et *dynamiquement*. Car c'est à son lancement que le programme charge les *plugins* : il va rechercher ceux implémentant les interfaces attendues. En effet, le code de la bibliothèque est inconnu car compilé, seul sont connues les fonctions attendues du *plugin*.

Un exemple de *plugins* sont typiquement les extensions d'un navigateur web. Ces composants ont été écrits pour le navigateur, en répondant à une interface attendue par ce logiciel, et peuvent être activés ou désactivés à volonté par l'utilisateur.
L'intérêt d'une telle approche est qu'elle permet d'introduire de la modularité dans l'architecture du projet, les *plugins* pouvant être développés séparément du logiciel client. Et surtout, ils peuvent être distribués séparément du logiciel ou être remplacés au besoin sans remplacer tout le logiciel. Cela peut présenter une difficulté d'analyse : quoi externaliser dans des plugins ? et sous quelles interfaces ? Car si les interfaces changent, le composant doit obligatoirement être remplacé et mis-à-jour, brisant la compatibilité des anciennes versions.[42]

Concrètement, Vahana VR définit trois interfaces à implémenter pour les *plugins* :

- Une interface pour créer une entrée.
- Une interface pour créer une sortie.
- Une interface *discovery* dont le rôle est d'indiquer les possibilités de créations entrées/sorties.

Ainsi, lors de son exécution, Vahana VR va chercher les bibliothèques présentes dans le dossier `plugins` à sa racine et va charger les *plugins* implémentant au moins un de ces interfaces. Une fois chargés, le logiciel possède désormais une collection d'entrées ou de

5. Présentées dans la section Intégration des dépendances du VideoStitch Player (section 3.2.1 p.22).

sorties potentielles. En fonction de données renvoyée par les *discovery*, l'utilisateur peut demander, dans l'interface de Vahana VR, de créer une nouvelle entrée – ou sortie – pour *plugin*. Et via les interfaces implémentées par le *plugin*, le logiciel va pouvoir faire exécuter le code de création de l'entrée – ou sortie – du *plugin*.

4.2.2 Conception des *plugins*

Cinq *plugins* furent écrits et maintenus pour répondre aux besoins. Ils s'appuient tous sur du matériel, comme des caméras ou des cartes d'acquisitions, conçu par des entreprises externes à VideoStitch. L'approche retenue a donc été de créer un nouveau *plugin* par constructeur :

— Entrée/sortie SDI et HDMI avec les cartes d'acquisition DeckLink [6].
— Entrée HDMI avec les cartes d'acquisition Yuan [7].
— Entrée HDMI avec les cartes d'acquisition Magewell [8].
— Entrée PCIe avec les caméras xiB de Ximea [9].
— Entrée Ethernet GigE [10] avec les caméras Genie TS de Teledyne Dalsa [11].

Leurs développements respectifs ont suivi les même étapes :

Prise en main

Tout d'abord, il s'agissait d'installer la caméra ou la carte d'acquisition sur la machine de développement et de la mettre en fonctionnement. Le constructeur va fournir deux ressources : le pilote pour que le système puisse utiliser le matériel, et un SDK pour qu'un développeur puisse écrire une application utilisant ce matériel. Ce SDK est donc une dépendance du plugin : ses *headers* et bibliothèques ont été ajoutés aux dépendances sur le dépôt VideoStitch-deps.
La figure 4.3 résume les interactions de Vahana VR au matériel d'entrée/sortie : les *plugins* correspondent au patron de conception [12] *adaptateur*, c'est-à-dire qu'ils convertissent les interfaces et le fonctionnement du SDK du constructeur sous d'autres interfaces, celles attendues par le programme client, Vahana VR[45].

6. https://www.blackmagicdesign.com/fr/products/decklink/models
7. http://www.yuan.com.tw/en/products/capture/capture_mod.htm
8. http://www.magewell.com/hardware?lang=en
9. http://www.ximea.com/en/products/application-specific-oem-and-custom-cameras/pci-express-high-speed-cameras
10. Standard pour les caméras industrielle utilisant la transmission par Ethernet[44].
11. http://www.teledynedalsa.com/imaging/products/cameras/area-scan/genie-ts/
12. Ou *design pattern*, qui est une bonne pratique de conception logicielle.

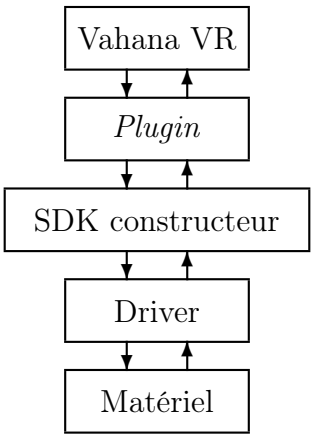

FIGURE 4.3 – Schéma des interactions de Vahana VR au matériel d'entrées/sorties

Implémentations

Ces interfaces attendues sont définies dans Vahana, et développer un *plugin* consiste à en réaliser les implémentations : ce fut la seconde partie du développement.
Tous les *plugins* développés implémentent l'interface `Input` et ses fonctions :

- `ing handle(const Config* parameters)` : indique si le plugin peut prendre en charge ces paramètres pour créer l'entrée.
- `Input* create(const Config* parameters)` : appellée si `handle` a réussi, cette fonction tente la création de l'entrée avec les paramètres donnés et renvoie un pointeur sur l'objet `Input` créé si cela a réussi, sinon un pointeur nul.
- `bool readFrame(uint32_t* frame)` : cette fonction renvoie la dernière image capturée par la caméra – ou carte d'acquisition –. Elle est appellée autant de fois que précisé dans les paramètres de la configuration, généralement au moins 25 fois par seconde pour obtenir une image fluide.
  L'image doit être renvoyée dans le format de couleur RGBA.

L'implémentation de l'interface `Output` est similaire à l'interface `Input`, à la différence qu'elle envoie des images, avec la méthode `writeFrame`, à l'instar de la méthode `readFrame`.

La création d'un `Input` réclame donc quelques paramètres et leurs valeurs, envoyés au travers de l'argument `parameters`. Cette configuration fournie par l'utilisateur contient, par exemple :

- Le nom du plugin cherché.
- Quelle caméra du plugin utiliser.
- La résolution désirée de la vidéo à capturer (largeur et hauteur en pixel).
- Le *framerate*, c'est-à-dire le nombre d'images par seconde qui seront capturées. C'est ce paramètre qui règle le nombre d'appels par secondes que Vahana VR va faire de la fonction `readFrame`.
- Le *pixel format* utilisé, c'est-à-dire le format de couleur. Un certain nombre d'encodages des couleurs pour un pixel existent et une sortie peut attendre un format

bien particulier, tout comme une entrée peut retourner des images sous un format précis.

Cette liste est enrichie par d'autres paramètres selon chaque *plugin* et les capacités du SDK du constructeur. Par exemple, Ximea propose des réglages de l'objectif de la caméra.

L'interface `Discovery` permet d'indiquer à Vahana VR, et par extension à l'utilisateur, les ensembles de paramètres et valeurs utilisables pour la création des entrées/sorties du *plugin*. L'implémentation de cette interface s'effectue avec quelques fonctions :

- `std::string name()` : indique le nom du plugin, qui sera utilisé dans par les paramètres de configuration.
- `std::vector<Device> devices()` : indique la liste des entrées/sorties disponibles : pour les cartes d'acquision cela correspond à l'ensemble des ports entrées/sorties présents sur la carte ; quant aux caméras, seules celles branchées apparraîtront. Un `Device` est simplement le numéro de la caméra, ou du port, ainsi que son type : entrée, sortie, ou les deux.
- `std::vector<DisplayMode> supportedDisplayModes(const Device& device)` : retourne l'ensemble des couples résolution-*framerate* supportés par le *device*.
- `std::vector<PixelFormats> supportedPixelFormat(const Device& device)` : retourne l'ensemble des formats de couleur supportés par le *device*.

La documentation accompagnant les SDK est ensuite lue, tout comme les programmes examples fournis, pour comprendre les possiblités offertes, définir les différents paramètres qui seront utilisés, et, enfin, implémenter les fonctions des interfaces.
Un résumé global de ces implémentations peut être décrit, les approches des différents SDK restant globalement les mêmes. Par exemple, voici l'implémentation des méthodes `create(const Config* parameters)` et `devices()`, dont les fonctionnements sont assez similaires :

1. Tout d'abord, une initialisation du SDK du constructeur est réalisée. Si elle échoue, le driver n'est alors pas installé.
2. Seulement pour `create` sont récupérés l'ensemble des paramètres de la configuration et la validité de leurs valeurs est testée.
3. Une itération est réalisée sur tous les *devices* disponibles.
   `create` va sélectionner celui précisé dans les paramètres, quand `devices` va tous les garder en mémoire.
4. Un pointeur [13] est récupéré sur chaque *device* sélectionné à l'étape précédente.
5. Grace à ce pointeur, le *device* est configuré : selon les paramètres pour `create` ou afin de détecter les paramètres supportés pour `devices`.
6. Seulement pour `create` est lancée la capture vidéo.

13. Ou encore appelé *handle* dans les SDK utilisant le système COM de Microsoft.

7. Enfin la méthode se termine et retourne un pointeur sur l'objet implémentant l'interface `Input` pour `create` ; ou la liste des `devices` pour la fonction `devices`.

Si une de ces étapes échoue, la méthode s'arrête à son tour et écrit un message d'erreur qui pourra être récupéré et affiché par Vahana VR à l'utilisateur.

De même, la fonction `readFrame` consiste toujours en deux étapes :

1. Lire la dernière image capturée par l'entrée.
2. Transformer l'image du format de couleur de capture vers le format de couleur RGBA. En effet, pour optimiser les algorithmes, les applications VideoStitch ne travaillent qu'avec ce format. Ainsi un certain nombre de fonctions, utilisant CUDA [14], sont disponibles pour opérer ces transformations vers le RGBA. La méthode appelle simplement la fonction nécessaire et en copie le résultat dans le tampon `uint32_t* frame` pour Vahana VR.

4.2.3 Quelques problèmes et solutions spécifiques

Chaque *plugin* s'appuyant sur un SDK d'un constructeur différent, un certain nombre de problèmes furent rencontrés pour réaliser l'implémentation totale des interfaces. Quelques difficultés et leurs solutions sont notables.

Fonctions de rappel

La fonction `readFrame` a souvent été mise en œuvre par un système de fonctions de rappels : bien souvent pour récupérer les images, il fallait passer au SDK lors de la configuration de l'entrée un pointeur sur une fonction[46]. Ainsi à chaque image capturée par la caméra ou le port, cette fonction de rappel sera appellée par le SDK avec le contenu de l'image en argument. Ce contenu est donc écrit dans une variable du plugin qui sera lue par la méthode `readFrame`.
Dès lors, il peut se créer un problème de synchronisation entre ces deux fonctions. La ressource partagée doit être protégée d'une double utilisation : une exclusion mutuelle[47] a été mise en place. De plus, la ressource ne doit être écrite qu'une fois qu'elle a été lue : les deux fonctions ont donc été placées dans des threads séparés et une condition variable a été utilisée. C'est une primitive de synchronisation C++ qui permet de bloquer deux threads jusqu'à l'un reçoive une notification de l'autre[48] : que la variable a été lue pour la fonction de rappel, et que la variable a été écrite pour la fonction `readFrame`.

14. Voir la section VideoStitch SDK (section 1.3.4 p.13).

Partage des ressources Yuan

Augmentation des interfaces DeckLink

Lecture des images Ximea

4.3 Déploiement

4.3.1 Création du dépôt VideoStitch-IO

Les *plugins* étaient à l'origine développés dans le dépôt VideoStitch-apps. Il faisait cependant plus sens qu'ils soient déplacés dans un dépôt distinct, car étant relativement indépendant de Vahana VR. Cette opération nécessitait, comme dans la section Intégration au *workflow* du VideoStitch Player et de Vahana VR (section 3.2.2 p.23), un déplacement du dossier de développement et de l'historique des modifications. La technique présentée en annexe A (p.41) fut de nouveau utilisée avec succès, déplaçant le contenu du dossier `src/plugins/` du dépôt VideoStitch-apps vers le dossier `src/` sur le dépôt VideoStitch-IO.

Ce nouveau dépôt a été organisé de manière sensiblement équivalente à celui de VideoStitch-apps, cependant la chaîne de compilation des *plugins* fut mise à jour pour prendre en compte cette nouvelle organisation.
En outre, ce dépôt nécessitait, tout comme VideoStitch-apps, du VideoStitch-SDK et des dépendances pour être compilé. Le script `update.sh`[15] fut alors partagé, et les *plugins* purent être compilés à nouveau.

Le Buildbot fut alors mis à jour également d'un nouveau builder prenant en charge ce dépôt. Une liste d'étapes similaires au listing 6 (p.27) furent écrites pour compiler automatiquement les *plugins* et les mettre à disposition sur une page de téléchargement interne.
Le script `update.sh` a alors été augmenté pour télécharger la dernière version des *plugins* et les copier dans la copie du dépôt VideoStitch-apps du développeur appelant le script. Les *plugins* ont été déplacés sur un autre dépôt mais sont toujours nécessaires pour générer utiliser Vahana VR avec toutes ses capacités.
Enfin, les *plugins* DeckLink, Magewell et Yuan furent ajoutés à l'installateur de Vahana VR, permettant de couvrir les besoins usuels du logiciel et les caméras les plus répandues sur le marché : les entrées/sorties SDI et HDMI.

4.3.2 Documentation

Une documentation fut écrite pour chaque *plugin*, décrivant essentiellement leur usage aux développeurs et aux utilisateurs avancés : l'interface de Vahana VR devrait normale-

15. Voir la section Intégration au *workflow* du VideoStitch Player et de Vahana VR (section 3.2.2 p.23).

ment suffire à créer les entrées et sorties. Ces documentations reviennent sur l'installation des drivers, quel SDK et quelle version a été utilisée, et quels paramètres, avec leurs valeurs possibles, sont utilisés par le plugin.
Ces documentations sont toutes rappatriées et fusionnées à celle de Vahana VR qui est ajoutée à l'installateur.

En outre, une documentation générale fut écrite sur le wiki interne. Cette version revient plus spécifiquement sur chaque plugin : quels interfaces sont implémentées, comment et pourquoi. Un état de l'avancement et de fonctionnement de chaque plugin a été tenu à jour.

4.3.3 Tests et *QA* de l'intégration des *plugins*

En parallèle au développement des plugins, l'équipe travailla à leur intégration dans Vahana VR grâce aux interfaces, qui étaient déjà connues. L'utilisation de plugins se révéla particulièrement pratique : il suffisait de remplacer les bibliothèques par de nouvelles versions pour obtenir les nouvelles capacitées implémentées.

Ce travail en parallèle fut très utile, car l'équipe de développement étant cliente de ses plugins, de nombreux tests furent effectués pour s'assurer d'une part de la bonne implémentation des plugins, et d'autre part de leur bonne intégration dans Vahana VR. Combiné à l'utilisation des pratiques agiles, les tickets de cette mission ont vu un certains nombre d'aller-retour entre leur *QA* par l'équipe et leur développement, que ce soit pour des bogues ou des cas d'usages non prévus.
Pour achever cet audit qualité, de nombreuses démonstrations complète ont été faites sur des machines de démonstration vierge d'environnement de développement, et quelques clients, dits *early adopter*, ont pu avoir accès aux plugins, alors encore en développement, notamment pour les utiliser sur des caméras, cartes d'acquisition ou écrans (pour récupérer les sorties) n'ayant pas été testés en interne. Pour exemple, il est arrivé à deux reprise d'organiser une journée de travail en conversation avec un client pour obtenir une version fonctionnelle à ses besoins. D'après ses retours, le plugin était corrigé et renvoyé compilé afin qu'il le remplace simplement dans son dossier d'installation et le teste à nouveau.
Par ces pratiques, les plugins ont au final gagné en qualité et stabilité.

Enfin, les deux plugins Teledyne Dalsa et Ximea ont fait l'objet de démonstrations vidéo dans Vahana VR pour le client, pour montrer les fonctionnalités et performances du logiciel et ses capacités d'intégration de nouvelles caméras. La dernière semaine du stage eut lieu une démonstration en direct en présence du client de l'utilisation des caméras Ximea dans Vahana VR.

4.4 Bilan et suite

Cette mission a permis d'apporter, sous forme de modules utilisables par Vahana VR, un certain nombre de capacités entrées/sorties au logiciel principalement sur des standards vidéos comme le SDI et le HDMI. D'autres standards ont pu être supportés tout aussi facilement pour le compte d'un client.
L'intégration de ces plugins a pu être réalisée en parallèle de cette mission, ce qui a été un gain de rapidité de développement mais aussi un gain de stabilité important : l'architecture modulaire permet de n'utiliser que les plugins désirés, sous la version voulue.
C'est également pourquoi ces plugins ont été externalisés dans un dépôt différent : leur cycle de développement est indépendant du logiciel. En outre, Vahana VR étant client de ces composants, des tests importants et utiles de qualité ont pu être menés sur ces plugins. Une grande partie du succès de ce logiciel qui sortira prochainnement tiendra à sa capacité à supporter de nombreuses entrées/sorties vidéo.

Il reste cependant du travail à effectuer : les interfaces entrées/sorties devraient être augmentées pour prendre en charge l'audio, et l'interface *discovery* devrait intégrer une détection de signal des paramètres actuellement configurés sur les caméras branchées en entrée ou sur les écrans en sorties.
Par ailleurs l'architecture de cette solution est une grande qualité, car de nouveaux plugins pourront être développés très facilement : il pourrait être très intéressant d'ouvrir les sources de ce dépôt aux clients afin qu'ils puissent développer le support de leur propre matériel dans l'ecosystème de Vahana VR. Une interface de programmation [16] pourrait être même plus généralement développée pour l'intégralité du logiciel, pour, à l'instar de toutes les grandes entreprises du web[49], développer de nouveaux marchés et tisser de nombreuses interactions avec d'autres écosystèmes[50][51], chacun devenant composant d'un système d'innovation et de croissance inter-logiciels.

16. Ou API pour *Application Programming Interface*, est une interface par laquelle un logiciel offre des services à un autre logiciel[49].

5. Conclusion

> *« Il semble que la perfection soit atteinte, non quand il n'y a plus rien à ajouter mais quand il n'y a plus rien à retrancher »*
>
> — Saint-Exupéry, Terre des hommes

Ces 7 mois de stage en tant qu'assistant ingénieur chez VideoStitch m'ont beaucoup plu et ont répondus à mes attentes.
Je souhaitais travailler dans une start-up et ce pour plusieurs raisons : tout d'abord je souhaitais avoir un aperçu de la chaîne de développement complète au sein d'une équipe regroupant différentes compétences des métiers de l'ingénieur. Ma première mission m'a permis de travailler sur le *workflow* de l'équipe de développement et sur l'architecture des logiciels de VideoStitch, et ainsi de comprendre cette chaîne de bout en bout.
Je souhaitais également intégrer une entreprise innovant de nouveaux usages numériques. Ma seconde mission m'a pleinement convaincu, en travaillant sur le système de *plugins*. Passionné par le domaine de la photographie j'ai pu travailler avec de nombreuses caméras sur un produit d'imagerie numérique et de vision par ordinateur.

La méthodologie Scrum fut très intéressante et avoir pu l'expérimenter me confirme dans sa pertinence de gestion des projets logiciels. Je compte utiliser des méthodes agiles pour la gestion de mes futurs projets .
Outre la méthodologie, la dynamique et la petite taille de l'équipe au sein de l'entreprise a été très agréable et stimulante : la communication est spontanée et très régulière, utilisant les supports écrits comme oraux. C'est finalement l'équipe plus que les personnes qui apprend et progresse.
Quant au contenu du stage, sa richesse et la complexité du domaine de l'entreprise m'a permis d'apprendre énormément sur le développement et l'architecture logiciel en C++ et en Qt. Outre les aspects techniques poussés et les nombreux concepts de programmation et autres bonnes pratiques, je retiendrai particulièrement la citation inscrite au début de cette conclusion. Je la dois à Nicolas Lopez, suite à une soirée relativement ardue de déboguage pour le compte d'un client : *Keep It Simple and Short*.

A l'issu de ce stage, j'ai donc décidé de suivre la filière *Ingénierie des Connaissances et des Supports d'Information*, pour faire suite à ce stage.
Enfin, j'espère avoir fourni à VideoStitch un travail à la hauteur des besoins exprimés et c'est avec attention que je suivrai la sortie et l'évolution de Vahana VR.

A. Déplacer des fichiers entre des dépôts Git en préservant l'historique [1]

L'objectif est de déplacer un dossier d'un dépôt A vers un dépôt B, tout en conservant l'historique de ses modifications. Cependant le dépôt B peut déjà contenir des fichiers et un historique. L'opération se déroule en deux étapes.

Premièrement il faut préparer le dossier du dépôt A, comme le montre le listing 9. Après une création d'une nouvelle copie locale du dépôt A avec `git clone` (ligne 1), la commande `git filter-branch` va supprimer tous les fichiers et toutes les modifications qui ne concerne pas le dossier (ligne 2). Le dépôt contient désormais à sa racine le contenu de ce dossier. Il faut le déplacer vers le nom du dossier souhaité pour le dépôt B (ligne 3) puis enregistrer cette modification avec `git commit` (ligne 4).

```
1    git clone <repository A url> && cd <repository A url>
2    git filter-branch --subdirectory-filter <the directory> -- --all
3    mkdir <the directory> && mv -k * <the directory>
4    git add . && git commit
```

Listing 9 – Préparation du dossier du dépôt A

Ensuite, il s'agit de déplacer le dossier tel que le montre le listing 10. Après avoir créé une copie du dépôt B (ligne 1), la gestion décentralisée des dépôts par Git permet de créer une *connexion à la copie locale* du dépôt A (ligne 2) et d'en tirer fichiers et historique (ligne 3). Le dossier a été déplacé, la copie du dépôt A peut être supprimée (ligne 4).

```
1    git clone <repository B url> && cd <repository B url>
2    git remote add link-to-repo-A ../<repository A>
3    git pull link-to-repo-A
4    git remove rm link-to-repo-A && rm -r ../<repository A>
```

Listing 10 – Déplacement du dossier du dépôt A au dépôt B

1. Cette technique est inspirée d'une question sur le site stackoverflow dont voici le lien : http://stackoverflow.com/q/1365541.

Bibliographie

[1] Scott Kelby. Create panoramic images with Photomerge, décembre 2010. http://helpx.adobe.com/photoshop/using/create-panoramic-images-photomerge.html. License CC BY-NC-SA 3.0.

[2] New House Internet Services. Create high quality panoramic images. http://www.ptgui.com/.

[3] By Noso1 at en.wikipedia.org from Wikimedia Commons. File :Rochester NY.jpg, novembre 2010. http://commons.wikimedia.org/wiki/File:Rochester_NY.jpg.

[4] Projection cartographique, janvier 2015. https://fr.wikipedia.org/wiki/Projection_cartographique. License CC BY-SA 3.0.

[5] Nicolas Burtey. What is equirectangular ?, octobre 2014. http://support.video-stitch.com/hc/en-us/articles/203657036-What-is-equirectangular-.

[6] Traroth from Wikimedia Commons. File :Projection cylindrique.jpg, mars 2005. https://commons.wikimedia.org/wiki/File:Projection_cylindrique.jpg. License CC BY-SA 3.0.

[7] modified by Mdf via Wikimedia Commons Original by NASA (sensor Terra/MODIS). File :Equirectangular-projection.jpg, 2002 (original), 2006 (modification by Mdf). https://commons.wikimedia.org/wiki/File:Equirectangular-projection.jpg. Public domain.

[8] réalité virtuelle, février 2014. https://fr.wiktionary.org/wiki/réalité_virtuelle.

[9] Jaunt. https://www.crunchbase.com/organization/jaunt.

[10] Mark Zuckerberg, mars 2014. https://www.facebook.com/zuck/posts/10101319050523971.

[11] Samsung Gear VR, décembre 2014. http://www.samsung.com/global/microsite/gearvr.

[12] Microsoft HoloLens | Official Site, décembre 2014. http://www.microsoft.com/microsoft-hololens.

[13] Google Cardboard, juin 2014. https://www.google.com/get/cardboard.

[14] Nicolas Burtey. What are the different products of VideoStitch ?, octobre 2014. http://support.video-stitch.com/hc/en-us/articles/203885163-What-are-the-different-products-of-VideoStitch-.

[15] Romain Bouqueau. GPU & CUDA : technical insights of a panoramic stitcher. http://www.video-stitch.com/gpu-cuda-technical-insights/.

[16] Nils Duval. What's up with stereographic 3D panoramic video ? http://www.video-stitch.com/whats-stereographic-3d-panoramic-video/.

[17] Image stéréoscopique, décembre 2011. https://fr.wikipedia.org/wiki/Image_stéréoscopique. License CC BY-SA 3.0.

[18] Méthode agile, janvier 2015. https://fr.wikipedia.org/wiki/Méthode_agile. License CC BY-SA 3.0.

[19] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. juillet 2013. http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf.

[20] Lakework from Wikimedia Commons. File :Scrum process.svg, janvier 2009. https://commons.wikimedia.org/wiki/File:Scrum_process.svg. License CC BY-SA 3.0.

[21] Qt, décembre 2014. https://fr.wikipedia.org/wiki/Qt. License CC BY-SA 3.0.

[22] Gestion de versions, octobre 2014. https://fr.wikipedia.org/wiki/Gestion_de_versions. License CC BY-SA 3.0.

[23] Continuous integration, janvier 2015. https://fr.wikipedia.org/wiki/Continuous_integration. License CC BY-SA 3.0.

[24] Scott Chacon and Ben Straub. *Pro Git*. Second Edition.

[25] Buildbot Operating Comitee. Buildbot. http://buildbot.net. GNU Public License version 2.

[26] Workflow, juillet 2014. https://fr.wikipedia.org/wiki/Workflow. License CC BY-SA 3.0.

[27] Software build, novembre 2014. https://en.wikipedia.org/wiki/Software_build. License CC BY-SA 3.0.

[28] Build automation, août 2014. https://en.wikipedia.org/wiki/Build_automation. License CC BY-SA 3.0.

[29] Chaîne de compilation, septembre 2013. https://fr.wikipedia.org/wiki/Chaîne_de_compilation. License CC BY-SA 3.0.

[30] Dépendance logicielle, septembre 2013. https://fr.wikipedia.org/wiki/Dépendance_logicielle. License CC BY-SA 3.0.

[31] Bibliothèque logicielle, août 2014. https://fr.wikipedia.org/wiki/Bibliothèque_logicielle. License CC BY-SA 3.0.

[32] 32-bit vs 64-bit, janvier 2015. https://en.wikipedia.org/wiki/64-bit_computing#32-bit_vs_64-bit. License CC BY-SA 3.0.

[33] Comment : définir des configurations Debug et Release. https://msdn.microsoft.com/fr-fr/library/wx0123s5.aspx.

[34] VideoLAN. libVLC. http://www.videolan.org/vlc/libvlc.html.

[35] Oculus VR, LLC. Oculus Developer Guide, décembre 2014. http://static.oculus.com/sdk-downloads/documents/Oculus_Developer_Guide_0.4.4.pdf.

[36] Logiciel, janvier 2015. https://fr.wikipedia.org/wiki/Logiciel. License CC BY-SA 3.0.

[37] Oculus VR, LLC. DK2s Now Shipping, juillet 2014. https://www.oculus.com/blog/dk2s-now-shipping-new-0-4-0-sdk-beta-and-comic-con/.

[38] Pilote informatique, janvier 2015. https://fr.wikipedia.org/wiki/Pilote_informatique. License CC BY-SA 3.0.

[39] High-Definition Multimedia Interface, janvier 2015. https://fr.wikipedia.org/wiki/High-Definition_Multimedia_Interface. License CC BY-SA 3.0.

[40] Serial Digital Interface, mars 2013. https://fr.wikipedia.org/wiki/Serial_Digital_Interface. License CC BY-SA 3.0.

[41] PCI Express, janvier 2015. https://fr.wikipedia.org/wiki/PCI_Express. License CC BY-SA 3.0.

[42] Programmation orientée composant, décembre 2014. https://fr.wikipedia.org/wiki/Programmation_orientée_composant. License CC BY-SA 3.0.

[43] Plugin, décembre 2014. https://fr.wikipedia.org/wiki/Plugin. License CC BY-SA 3.0.

[44] GigE Vision, novembre 2014. https://fr.wikipedia.org/wiki/GigE_Vision. License CC BY-SA 3.0.

[45] Alexander Shvets, Gerhard Frey, and Marina Pavlova. Adapter Design Pattern. http://sourcemaking.com/design_patterns/adapter. License CC BY-NC-ND 3.0.

[46] Fonction de rappel, décembre 2014. https://fr.wikipedia.org/wiki/Fonction_de_rappel. License CC BY-SA 3.0.

[47] Exclusion mutuelle, mars 2014. https://fr.wikipedia.org/wiki/Exclusion_mutuelle. License CC BY-SA 3.0.

[48] std : :condition_variable, octobre 2014. http://en.cppreference.com/w/cpp/thread/condition_variable. License CC BY-SA 3.0.

[49] Interface de programmation, septembre 2014. https://fr.wikipedia.org/wiki/Interface_de_programmation. License CC BY-SA 3.0.

[50] InternetActu.net et la Fing. La valeur stratégique des API - Harvard Business Review, janvier 2015. http://alireailleurs.tumblr.com/post/109469055374/la-valeur-strategique-des-api-harvard-business.

[51] Harvard Business Review. The Strategic Value of APIs, janvier 2015. https://hbr.org/2015/01/the-strategic-value-of-apis.

Table des figures